Programming Challenge: Three Card Flush

Abstract: The first "lab" of this class. This lengthy program creates a card game that can be played against a computer or with a friend. It takes statistical analysis of the games as well.

Task 1: The Deck and the Shuffle

Task 1 Demo:

```
[3]> ( demo--make-deck )
>>> Testing: make-deck
--- deck =
((2 . CLUB) (3 . CLUB) (4 . CLUB) (5 . CLUB) (6 . CLUB) (7 . CLUB) (8 . CLUB) (9 . CLUB) (10 . CLUB) (JACK . CLUB) (QUEEN . CLUB) (KING . CLUB) (ACE . CLUB)
 (2 . DIAMOND) (3 . DIAMOND) (4 . DIAMOND) (5 . DIAMOND) (6 . DIAMOND) (7 . DIAMOND) (8 . DIAMOND) (9 . DIAMOND) (10 . DIAMOND) (JACK . DIAMOND) (QUEEN . DIAMOND)
 (KING . DIAMOND) (ACE . DIAMOND) (2 . HEART) (3 . HEART) (4 . HEART) (5 . HEART) (6 . HEART) (7 . HEART) (8 . HEART) (9 . HEART) (10 . HEART) (JACK . HEART)
 (QUEEN . HEART) (KING . HEART) (ACE . HEART) (2 . SPADE) (3 . SPADE) (4 . SPADE) (5 . SPADE) (6 . SPADE) (7 . SPADE) (8 . SPADE) (9 . SPADE) (10 . SPADE) (JACK . SPADE)
 (QUEEN . SPADE) (KING . SPADE) (ACE . SPADE))
--- number of cards in deck = 52
NIL
[4]> ( demo--establish-shuffled-deck )
>>> Testing: shuffle-deck
--- *deck* ...
((QUEEN . SPADE) (ACE . CLUB) (JACK . DIAMOND) (10 . CLUB) (ACE . DIAMOND) (KING . DIAMOND) (7 . CLUB) (3 . SPADE) (6 . CLUB) (2 . DIAMOND) (6 . DIAMOND) (3 . CLUB)
 (3 . HEART) (ACE . HEART) (10 . DIAMOND) (JACK . SPADE) (ACE . SPADE) (JACK . HEART) (9 . CLUB) (7 . HEART) (9 . DIAMOND) (6 . SPADE) (10 . HEART) (4 . DIAMOND)
 (2 . CLUB) (QUEEN . HEART) (5 . DIAMOND) (5 . CLUB) (2 . SPADE) (5 . SPADE) (8 . HEART) (4 . SPADE) (QUEEN . CLUB) (4 . CLUB) (KING . CLUB) (8 . CLUB) (9 . HEART)
 (JACK . CLUB) (10 . SPADE) (9 . SPADE) (3 . DIAMOND) (7 . SPADE) (6 . HEART) (5 . HEART) (QUEEN . DIAMOND) (8 . DIAMOND) (KING . SPADE) (8 . SPADE) (KING . HEART)
 (7 . DIAMOND) (2 . HEART) (4 . HEART))
--- number of cards in *deck* = 52
NIL
```

Task 1 Code:

```
( defun make-deck ()
    ( mapcan #'make-selected-deck '( club diamond heart spade ) )
)

( setf *value* '( 2 3 4 5 6 7 8 9 10 jack queen king ace ) )

( defun make-selected-deck ( suite &aux rank )
    ( setf suite-duplicates ( duplicate ( length *value* ) suite ) )
    ( mapcar #'cons *value* suite-duplicates )
)

( defun demo--make-deck ()
    ( format t ">>> Testing: make-deck~%" )
    ( setf deck ( make-deck ) )
    ( format t "--- deck = ~A~%" deck )
    ( format t "--- number of cards in deck = ~A~%" ( length deck ) )
    nil
)

( defun establish-shuffled-deck ()
    ( setf *deck* ( shuffle ( make-deck ) ) )
    nil
)
```

Task 1 Code cont:

```
( defun demo--establish-shuffled-deck ()
   ( format t ">>> Testing: shuffle-deck~%" )
   ( establish-shuffled-deck )
   ( format t "--- *deck* ... ~A~%" *deck* )
   ( format t "--- number of cards in *deck* = ~A~%" ( length *deck* ) )
   nil
)
```

Task 2: The Hands, the Deal and the Discard

Task 2 Demo:

```
[5]> ( demo--deal-hands )
>>> Testing: deal-hands
--- *hand1* = ((5 . SPADE) (9 . DIAMOND) (6 . CLUB))
--- *hand2* = ((6 . SPADE) (5 . CLUB) (ACE . DIAMOND))
--- number of cards in *deck* = 46
NIL
[6]> ( demo--deal-hands )
>>> Testing: deal-hands
--- *hand1* = ((QUEEN . HEART) (8 . HEART) (2 . HEART))
--- *hand2* = ((ACE . HEART) (JACK . SPADE) (7 . SPADE))
--- number of cards in *deck* = 46
NIL
[7]> ( demo--randomly-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((KING . HEART) (QUEEN . DIAMOND) (JACK . HEART))
--- *hand2* = ((2 . DIAMOND) (7 . DIAMOND) (5 . CLUB))
--- *hand1* = (NIL (QUEEN . DIAMOND) (JACK . HEART))
--- *hand2* = ((2 . DIAMOND) NIL (5 . CLUB))
NIL
[8]> ( demo--randomly-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((KING . SPADE) (3 . CLUB) (ACE . DIAMOND))
--- *hand2* = ((3 . SPADE) (4 . CLUB) (9 . CLUB))
--- *hand1* = (NIL (3 . CLUB) (ACE . DIAMOND))
--- *hand2* = ((3 . SPADE) NIL (9 . CLUB))
NIL
[9]> ( demo--randomly-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((KING . SPADE) (2 . SPADE) (ACE . CLUB))
--- *hand2* = ((ACE . HEART) (5 . DIAMOND) (QUEEN . HEART))
--- *hand1* = ((KING . SPADE) NIL (ACE . CLUB))
--- *hand2* = ((ACE . HEART) NIL (QUEEN . HEART))
NIL
```

Task 2 Demo cont:

```
[10]> ( demo--randomly-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((10 . DIAMOND) (9 . HEART) (KING . SPADE))
--- *hand2* = ((4 . SPADE) (6 . HEART) (5 . SPADE))
--- *hand1* = ((10 . DIAMOND) (9 . HEART) NIL)
--- *hand2* = ((4 . SPADE) (6 . HEART) NIL)
NIL
[11]> ( demo--randomly-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((KING . SPADE) (6 . CLUB) (8 . HEART))
--- *hand2* = ((KING . DIAMOND) (QUEEN . SPADE) (5 . CLUB))
--- *hand1* = (NIL (6 . CLUB) (8 . HEART))
--- *hand2* = ((KING . DIAMOND) (QUEEN . SPADE) NIL)
NIL
```

Task 2 Code:

```
( defun deal-hands ()
   ( establish-shuffled-deck )
   ( setf *hand1* () )
   ( setf *hand2* () )
   ( deal-card-to-hand1 )
   ( deal-card-to-hand2 )
   ( deal-card-to-hand1 )
   ( deal-card-to-hand2 )
   ( deal-card-to-hand1 )
   ( deal-card-to-hand2 )
   nil
)

( defun deal-card-to-hand1 ()
   ( setf *hand1* ( append *hand1* ( list ( car *deck* ) ) ) )
   ( setf *deck* ( remove ( car *deck* ) *deck* ) )
)

( defun deal-card-to-hand2 ()
   ( setf *hand2* ( append *hand2* ( list ( car *deck* ) ) ) )
   ( setf *deck* ( remove ( car *deck* ) *deck* ) )
)

( defun demo--deal-hands ()
   ( format t ">>> Testing: deal-hands~%" )
   ( deal-hands )
   ( format t "--- *hand1* = ~A~%" *hand1* )
   ( format t "--- *hand2* = ~A~%" *hand2* )
   ( format t "--- number of cards in *deck* = ~A~%" ( length *deck* ) )
   nil
)
```

Task 2 Code cont:

```
( defun randomly-discard-cards ()
   ( randomly-discard-card-from-hand1 )
   ( randomly-discard-card-from-hand2 )
   nil
)

( defun randomly-discard-card-from-hand1 ()
   ( setf ( nth ( random 3 ) *hand1*) nil)
)

( defun randomly-discard-card-from-hand2 ()
   ( setf ( nth ( random 3 ) *hand2*) nil)
)

( defun demo--randomly-discard-cards ()
   (format t ">>> Testing: randomly-discard-cards~%" )
   ( deal-hands )
   ( format t "--- *hand1* = ~A~%" *hand1* )
   ( format t "--- *hand2* = ~A~%" *hand2* )
   ( randomly-discard-cards )
   ( format t "--- *hand1* = ~A~%" *hand1* )
   ( format t "--- *hand2* = ~A~%" *hand2* )
   nil
)

( defun random-discard-help ( o l )
   ( cond
      ( ( null l ) nil )
      ( ( equal o ( car l ) ) ( cons nil ( cdr l ) ) )
      ( t
         ( cons ( car l ) ( random-discard-help o ( cdr l ) ) )
      )
   )
)
```

Task 3: Replacing Cards in Hands, Taking Turns

Task 3 Demo:

```
[12]> ( demo--replace-cards )
>>> Testing: replace-cards
--- *hand1* ((2 . CLUB) (8 . SPADE) (JACK . CLUB))
--- *hand2* ((8 . HEART) (7 . DIAMOND) (8 . CLUB))
--- *hand1* (NIL (8 . SPADE) (JACK . CLUB))
--- *hand2* (NIL (7 . DIAMOND) (8 . CLUB))
--- *hand1* ((4 . SPADE) (8 . SPADE) (JACK . CLUB))
--- *hand2* ((5 . CLUB) (7 . DIAMOND) (8 . CLUB))
NIL
[13]> ( demo--replace-cards )
>>> Testing: replace-cards
--- *hand1* ((5 . SPADE) (4 . CLUB) (8 . CLUB))
--- *hand2* ((QUEEN . SPADE) (JACK . HEART) (8 . HEART))
--- *hand1* ((5 . SPADE) (4 . CLUB) NIL)
--- *hand2* (NIL (JACK . HEART) (8 . HEART))
--- *hand1* ((5 . SPADE) (4 . CLUB) (JACK . DIAMOND))
--- *hand2* ((3 . HEART) (JACK . HEART) (8 . HEART))
NIL
[14]> ( demo--replace-cards)
>>> Testing: replace-cards
--- *hand1* ((10 . DIAMOND) (ACE . CLUB) (4 . DIAMOND))
--- *hand2* ((5 . DIAMOND) (7 . SPADE) (JACK . CLUB))
--- *hand1* (NIL (ACE . CLUB) (4 . DIAMOND))
--- *hand2* (NIL (7 . SPADE) (JACK . CLUB))
--- *hand1* ((4 . SPADE) (ACE . CLUB) (4 . DIAMOND))
--- *hand2* ((10 . SPADE) (7 . SPADE) (JACK . CLUB))
NIL
[15]> ( demo--replace-cards )
>>> Testing: replace-cards
--- *hand1* ((4 . SPADE) (10 . HEART) (9 . HEART))
--- *hand2* ((4 . HEART) (5 . HEART) (5 . DIAMOND))
--- *hand1* (NIL (10 . HEART) (9 . HEART))
--- *hand2* ((4 . HEART) (5 . HEART) NIL)
--- *hand1* ((8 . CLUB) (10 . HEART) (9 . HEART))
--- *hand2* ((4 . HEART) (5 . HEART) (2 . HEART))
NIL
```

```
[16]> ( demo--players-each-take-a-turn )
>>>Testing: players-each-take-a-turn
--- The hands ...
--- *hand1* = ((4 . CLUB) (10 . DIAMOND) (JACK . HEART))
--- *hand2* = ((3 . DIAMOND) (4 . SPADE) (QUEEN . DIAMOND))
--- Each players take a turn ...
--- *hand1* = ((QUEEN . CLUB) (10 . DIAMOND) (JACK . HEART))
--- *hand2* = ((3 . DIAMOND) (KING . HEART) (QUEEN . DIAMOND))
--- Each players take a turnn ...
--- *hand1* = ((QUEEN . CLUB) (6 . HEART) (JACK . HEART))
--- *hand2* = ((3 . DIAMOND) (KING . HEART) (9 . SPADE))
--- Each players take a turnn ...
--- *hand1* = ((7 . CLUB) (6 . HEART) (JACK . HEART))
--- *hand2* = ((3 . DIAMOND) (KING . HEART) (8 . HEART))
--- Each players take a turnn ...
--- *hand1* = ((7 . CLUB) (6 . DIAMOND) (JACK . HEART))
--- *hand2* = ((3 . DIAMOND) (5 . HEART) (8 . HEART))
NIL
```

Task 3 Code:

```
( defun replace-cards ()
   ( replace-cards-in-hand1 )
   ( replace-cards-in-hand2 )
)

( defun replace-cards-in-hand1 ()
   ( setf *hand1* ( replace-helper ( car *deck* ) *hand1* ) )
   ( setf *deck* ( cdr *deck* ) )
)

( defun replace-cards-in-hand2 ()
   ( setf *hand2* ( replace-helper ( car *deck* ) *hand2* ) )
   ( setf *deck* ( cdr *deck* ) )
)

( defun replace-helper ( o l )
   ( cond
      ( ( null l ) nil )
      ( ( equal nil ( car l ) ) ( cons o ( cdr l ) ) )
      ( t
         ( cons ( car l ) ( replace-helper o ( cdr l ) ) )
      )
   )
)

( defun demo--replace-cards ()
   ( format t ">>> Testing: replace-cards~%" )
   ( deal-hands )
   ( format t "--- *hand1* ~A~%" *hand1* )
   ( format t "--- *hand2* ~A~%" *hand2* )
   ( randomly-discard-cards )
   ( format t "--- *hand1* ~A~%" *hand1* )
   ( format t "--- *hand2* ~A~%" *hand2* )
   ( replace-cards )
   ( format t "--- *hand1* ~A~%" *hand1* )
   ( format t "--- *hand2* ~A~%" *hand2* )
   nil
)

( defun players-each-take-a-turn ()
   ( randomly-discard-cards )
   ( replace-cards )
   nil
)
```

Task 3 Code cont:

```
( defun demo--players-each-take-a-turn ()
   ( format t ">>>Testing: players-each-take-a-turn~%" )
   ( deal-hands )
   ( format t "--- The hands ... ~%" )
   ( format t "--- *hand1* = ~A~%" *hand1*  )
   ( format t "--- *hand2* = ~A~%" *hand2*  )
   ( players-each-take-a-turn )
   ( format t "--- Each players take a turn ...~%" )
   ( format t "--- *hand1* = ~A~%" *hand1*  )
   ( format t "--- *hand2* = ~A~%" *hand2*  )
   ( players-each-take-a-turn )
   ( format t "--- Each players take a turnn ...~%" )
   ( format t "--- *hand1* = ~A~%" *hand1*  )
   ( format t "--- *hand2* = ~A~%" *hand2*  )
   ( players-each-take-a-turn )
   ( format t "--- Each players take a turnn ...~%" )
   ( format t "--- *hand1* = ~A~%" *hand1*  )
   ( format t "--- *hand2* = ~A~%" *hand2*  )
   ( players-each-take-a-turn )
   ( format t "--- Each players take a turnn ...~%" )
   ( format t "--- *hand1* = ~A~%" *hand1*  )
   ( format t "--- *hand2* = ~A~%" *hand2*  )
   nil
)
```

Task 4: Hand Analysis

Task 4 Demo:

```
[17]> ( demo--flush-p )
>>> Testing: flush-p
((2 . CLUB) (ACE . CLUB) (10 . CLUB)) is a flush
((JACK . DIAMOND) (9 . DIAMOND) (5 . DIAMOND)) is a flush
((JACK . HEART) (10 . HEART) (9 . HEART)) is a flush
((2 . SPADE) (3 . SPADE) (ACE . SPADE)) is a flush
((10 . SPADE) (5 . DIAMOND) (ACE . SPADE)) is not a flush
((8 . CLUB) (9 . DIAMOND) (10 . HEART)) is not a flush
NIL
```

Task 4 Demo cont:

```
[18]> ( demo--high-card )
>>> Testing: high-card
(QUEEN . SPADE) is the high card of
 ((10 . HEART) (5 . CLUB) (QUEEN . SPADE) (7 . HEART))
(ACE . CLUB) is the high card of
 ((2 . DIAMOND) (2 . CLUB) (10 . HEART) (4 . DIAMOND) (ACE . CLUB))
(ACE . DIAMOND) is the high card of
 ((ACE . DIAMOND) (ACE . CLUB) (5 . SPADE))
NIL
[19]> ( demo--straight-p )
>>> Testing: straight-p
((5 . SPADE) (3 . DIAMOND) (4 . SPADE) (6 . CLUB)) is not a striaght
((5 . SPADE) (7 . DIAMOND) (4 . SPADE) (8 . CLUB)) is not a straight
((KING . HEART) (QUEEN . DIAMOND) (ACE . SPADE) (10 . CLUB) (JACK . DIAMOND)) is not a straight
((ACE . CLUB) (2 . DIAMOND) (3 . SPADE)) is not a straight
NIL
[20]> ( demo--analyze-hand )
>>> Testing: analyze-hand
((5 . SPADE) (3 . DIAMOND) (4 . SPADE)) is a BUST
((5 . CLUB) (9 . CLUB) (4 . CLUB)) is a (9 HIGH CLUB FLUSH)
((QUEEN . HEART) (ACE . HEART) (KING . HEART)) is a (ACE HIGH HEART FLUSH)
NIL
```

Task 4 Code:

```
( defun flush-p ( hand &aux suits )
   ( setf suits ( mapcar #'cdr hand ) )
   ( suits-checker suits )
)

( defun suits-checker ( li )
   ( cond
      ( ( < ( length li ) 2 ) t )
      ( ( eq ( car li ) ( cadr li ) ) ( suits-checker ( cdr li ) ) )
      (t
         nil )
   )
)
```

Task 4 Code cont:

```
( defun demo--flush-p ( &aux hand )
   ( format t ">>> Testing: flush-p~%" )
   ( setf hand '( ( 2 . club ) ( ace . club ) ( 10 . club ) ) )
   ( format t "~A " hand )
   ( if ( flush-p hand )
      ( format t "is a flush~%" )
      ( format t "is not a flush~%" )
   )
   ( setf hand '( ( jack . diamond ) ( 9 . diamond ) ( 5 . diamond ) ) )
   ( format t "~A " hand )
   ( if ( flush-p hand )
      ( format t "is a flush~%" )
      ( format t "is not a flush~%" )
   )
   ( setf hand '( ( jack . heart ) ( 10 . heart ) ( 9 . heart ) ) )
   ( format t "~A " hand )
   ( if ( flush-p hand )
      ( format t "is a flush~%" )
      ( format t "is not a flush~%" )
   )
   ( setf hand '( ( 2 . spade) ( 3 . spade ) ( ace . spade ) ) )
   ( format t "~A " hand )
   ( if ( flush-p hand )
      ( format t "is a flush~%" )
      ( format t "is not a flush~%" )
   )
   ( setf hand '( ( 10 . spade) ( 5 . diamond ) ( ace . spade ) ) )
   ( format t "~A " hand )
   ( if ( flush-p hand )
      ( format t "is a flush~%" )
      ( format t "is not a flush~%" )
   )
   ( setf hand '( ( 8 . club) ( 9 . diamond ) ( 10 . heart ) ) )
   ( format t "~A " hand )
   ( if ( flush-p hand )
      ( format t "is a flush~%" )
      ( format t "is not a flush~%" )
   )
)
```

Task 4 Code cont:

```
( defun high-card ( hand )
   ( setf handVal ( mapcar #'car hand ) )
   ( setf high-val ( higher-card handVal ) )
   ( setf high ( assoc high-val hand ) )
   high
)

( defun higher-card ( hand-val )
   ( setf firstVal ( position ( first hand-val ) *value* ) )
   ( setf secondVal ( position ( second hand-val ) *value* ) )
   ( cond
      ( ( = ( length hand-val ) 1 ) (car hand-val ) )
      ( ( > firstVal secondVal )
         ( higher-card ( cons ( car hand-val ) ( remove ( second hand-val ) hand-val ) ) )
      )
      (t
        ( higher-card ( cdr hand-val ) )
      )
   )
)

( defun demo--high-card ()
   ( format t ">>> Testing: high-card~%" )
   ( setf hand '( ( 10 . heart ) ( 5 . club ) ( queen . spade ) ( 7 . heart ) ) )
   ( format t "~A is the high card of~% ~A~%" ( high-card hand ) hand )
   ( setf hand '( ( 2 . diamond ) ( 2 . club ) ( 10 . heart ) ( 4 . diamond )
      ( ace . club ) ) )
   ( format t "~A is the high card of~% ~A~%" ( high-card hand ) hand )
   ( setf hand '( ( ace . diamond ) ( ace . club ) ( 5 . spade ) ) )
   ( format t "~A is the high card of~% ~A~%" ( high-card hand ) hand )
   nil
)


( defun straight-p ( hand )
   ( setf sorted-hand ( sort ( mapcar #'car hand ) #'value-sort ) )
   ( straight-check sorted-hand )
)

( defun straight-check ( cards )
   ( setf firstVal ( position ( car cards ) *value* ) )
   ( setf secondVal ( position ( car cards ) *value* ) )
   ( cond
      ( ( = ( length cards ) 1 ) t )
      ( ( not ( = ( abs ( - firstVal secondVal ) ) 1 ) ) nil )
      (t
         ( straight-check ( cdr cards ) )
      )
   )
)
```

Task 4 Code cont:

```
( defun value-sort ( v1 v2)
   ( setf firstVal ( position v1 *value* ) )
   ( setf secondVal ( position v2 *value* ) )
   ( < firstVal secondVal )
)

( defun demo--straight-p ()
   ( format t ">>> Testing: straight-p~%" )
   ( setf hand '( ( 5 . spade ) ( 3 . diamond ) ( 4 . spade ) ( 6 . club )) )
   ( format t "~A " hand )
   ( if ( straight-p hand )
      ( format t "is a straight~%" )
      ( format t "is not a striaght~%" )

   )
   ( setf hand '( ( 5 . spade ) ( 7 . diamond ) ( 4 . spade ) ( 8 . club ) ) )
   ( format t "~A " hand )
   ( if ( straight-p hand )
      ( format t " is a straight ~%" )
      ( format t "is not a straight~%" )
   )
   ( setf hand '( ( king . heart ) ( queen . diamond ) ( ace . spade ) ( 10 . club )
      ( jack . diamond ) ) )
      ( format t "~A " hand )
      ( if ( straight-p hand )
         ( format t "is a straight~%" )
         ( format t "is not a straight~%" )
      )
   ( setf hand '( ( ace . club ) ( 2 . diamond ) ( 3 . spade ) ) )
   ( format t "~A " hand )
   ( if ( straight-p hand )
      ( format t "is a straight~%" )
      ( format t "is not a straight~%" )
   )
   nil
)

( defun analyze-hand ( hand )
   ( cond
      ( ( not ( flush-p hand ) ) 'bust )
      ( t
         ( setf analysis '( flush ) )
         ( setf analysis ( cons ( cdr ( car hand ) ) analysis ) )
         ( if ( straight-p hand ) ( setf analysis ( cons 'straight analysis ) ) )
         ( setf analysis ( cons 'high analysis ) )
         ( setf analysis ( cons ( car ( high-card hand ) ) analysis ) )
         analysis
      )
   )
)
```

Task 4 Code cont:

```
( defun demo--analyze-hand ()
   ( format t ">>> Testing: analyze-hand~%" )
   ( setf hand '( ( 5 . spade) ( 3 . diamond ) ( 4 . spade ) ) )
   ( format t "~A is a ~A~%" hand ( analyze-hand hand ) )
   ( setf hand '( ( 5 . club) ( 9 . club ) ( 4 . club ) ) )
   ( format t "~A is a ~A~%" hand ( analyze-hand hand ) )
   ( setf hand '( ( queen . heart ) ( ace . heart ) ( king . heart ) ) )
   ( format t "~A is a ~A~%" hand ( analyze-hand hand ) )
   nil
)
```

Task 5 Game State and End of Game Reporting:

Task 5 Demo:

```
[21]> ( demo--analyze-game )
>>> Testing: analyze-game
Game 1 ...
*hand1* = ((2 . DIAMOND) (4 . DIAMOND) (JACK . HEART))
*hand2* = ((10 . SPADE) (KING . HEART) (QUEEN . HEART))
*game-state* = (BUST BUST)
*hand1* = ((10 . DIAMOND) (JACK . DIAMOND) (2 . DIAMOND))
*hand2* = ((3 . SPADE) (5 . SPADE) (4 . SPADE))
*game-state* = ((JACK HIGH DIAMOND FLUSH) (5 HIGH SPADE FLUSH))
NIL
[22]> ( demo--report-the-result )
>>> Testing: report-the-result - both bust
*hand1* = ((5 . DIAMOND) (2 . SPADE) (10 . HEART))
*hand2* = ((QUEEN . HEART) (4 . CLUB) (6 . DIAMOND))
*game-state* = (BUST BUST)
--> The game is a draw. The deck is dead.
>>> Testing: report-the-result - p1 flush, p2 bust
*hand1* = ((8 . HEART) (3 . HEART) (4 . HEART))
*hand2* = ((2 . CLUB) (KING . DIAMOND) (7 . SPADE))
*game-state* = ((8 HIGH HEART FLUSH) BUST)
--> Player 1 wins with (8 HIGH HEART FLUSH)
>>> Testing: report-the-result - p1 bust, p2 flush
*hand1* = ((6 . SPADE) (8 . HEART) (10 . CLUB))
*hand2* = ((6 . DIAMOND) (KING . DIAMOND) (4 . DIAMOND))
*game-state* = (BUST (KING HIGH DIAMOND FLUSH))
--> Player 2 wins with (KING HIGH DIAMOND FLUSH)
>>> Testing: report-the-result - p1 flush high, p2 flush
*hand1* = ((ACE . HEART) (2 . HEART) (6 . HEART))
*hand2* = ((4 . SPADE) (6 . SPADE) (9 . SPADE))
*game-state* = ((ACE HIGH HEART FLUSH) (9 HIGH SPADE FLUSH))
!!! Both players found their way to a flush
--> Player 1 wins with (ACE HIGH HEART FLUSH)
>>> Testing: report-the-result - p1 flush, p2 flush high
*hand1* = ((4 . SPADE) (8 . SPADE) (9 . SPADE))
*hand2* = ((7 . DIAMOND) (JACK . DIAMOND) (KING . DIAMOND))
*game-state* = ((9 HIGH SPADE FLUSH) (KING HIGH DIAMOND FLUSH))
!!! Both players found their way to a flush
--> Player 2 wins with (KING HIGH DIAMOND FLUSH)
>>> Testing: report-the-result - p1 stright flush, p2 flush
*hand1* = ((10 . DIAMOND) (JACK . DIAMOND) (QUEEN . DIAMOND))
*hand2* = ((JACK . HEART) (10 . HEART) (6 . HEART))
*game-state* = ((QUEEN HIGH DIAMOND FLUSH) (JACK HIGH HEART FLUSH))
!!! Both players found their way to a flush
--> Player 1 wins with (QUEEN HIGH DIAMOND FLUSH)
>>> Testing: report-the-result - p1 flush, p2 straight flush
*hand1* = ((6 . HEART) (2 . HEART) (8 . HEART))
*hand2* = ((9 . DIAMOND) (10 . DIAMOND) (JACK . DIAMOND))
*game-state* = ((8 HIGH HEART FLUSH) (JACK HIGH DIAMOND FLUSH))
!!! Both players found their way to a flush
--> Player 2 wins with (JACK HIGH DIAMOND FLUSH)
NIL
```

Task 5 Code:

```
( defun analyze-game ()
   ( setf *game-state* ( list ( analyze-hand *hand1* ) ( analyze-hand *hand2* ) ) )
)

( defun demo--analyze-game ()
   ( format t ">>> Testing: analyze-game~%" )
   ; a couple of busts
   ( format t "Game 1 ... ~%" )
   ( setf *hand1* '( ( 2 . diamond ) ( 4 . diamond ) ( jack . heart ) ) )
   ( setf *hand2* '( ( 10 . spade ) ( king . heart ) ( queen . heart ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   ; an ordinary flush and a straight flush( format t "Game 2 ... ~%" )
   ( setf *hand1* '( ( 10 . diamond ) ( jack . diamond ) ( 2 . diamond ) ) )
   ( setf *hand2* '( ( 3 . spade ) ( 5 . spade ) ( 4 . spade ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   nil
)
```

Task 5 Code cont:

```
( defun report-the-result ()
   ( cond
      ( ( equal *game-state* '( bust bust ) )
        ( increment '*draw-count* )
        ( format t "--> The game is a draw. The deck is dead.~%")
      )
      ( ( and ( not ( equal ( first *game-state* ) 'bust ) )
              ( equal ( second *game-state* ) 'bust )
        )
        ( increment '*win1-count* )
        ( format t "--> Player 1 wins with ~A~%" ( first *game-state* ) )
      )
      ( ( and ( equal ( first *game-state* ) 'bust )
              ( not ( equal ( second *game-state* ) 'bust ) )
        )
        ( increment '*win2-count* )
        ( format t "--> Player 2 wins with ~A~%" ( second *game-state* ) )
      )
      ( ( and ( straight-p *hand1* ) ( not ( straight-p *hand2* ) ) )
        ( increment '*win1-count* )
        ( increment '*f1f2-count* )
        ( format t "!!! Both players found their way to a flush~%" )
        ( format t "--> Player 1 wins with ~A~%" ( first *game-state* ) )
      )
      ( ( and ( not ( straight-p *hand1* ) ) ( straight-p *hand2* ) )
        ( increment '*win2-count* )
        ( increment '*f1f2-count* )
        ( format t "!!! Both players found their way to a flush~%" )
        ( format t "--> Player 2 wins with ~A~%" ( second *game-state* ) )
      )
      ( ( card-greater ( high-card *hand1* ) ( high-card *hand2* ) )
        ( increment '*f1f2-count* )
        ( increment '*win1-count* )
        ( format t "!!! Both players found their way to a flush~%" )
        ( format t "--> Player 1 wins with ~A~%" ( first *game-state* ) )
      )
      ( ( card-greater ( high-card *hand2* ) ( high-card *hand1* ) )
        ( increment '*f1f2-count* )
        ( increment '*win2-count* )
        ( format t "!!! Both players found their way to a flush~%" )
        ( format t "--> Player 2 wins with ~A~%" ( second *game-state* ) )
      )
   )
   nil
)

( defun card-greater ( card1 card2 )
   ( setf val1 ( position ( car card1 ) *value* ) )
   ( setf val2 ( position ( car card2 ) *value* ) )
   ( > val1 val2 )
)
```

Task 5 Code cont:

```
( defun demo--report-the-result ()
;both bust
   ( format t ">>> Testing: report-the-result - both bust~%")
   ( setf *hand1* '( ( 5 . diamond ) ( 2 . spade ) ( 10 . heart ) ) )
   ( setf *hand2* '( ( queen . heart ) ( 4 . club ) ( 6 . diamond ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   ( report-the-result )
;p1 flush - p2 bust
   ( format t ">>> Testing: report-the-result - p1 flush, p2 bust~%")
   ( setf *hand1* '( ( 8 . heart ) ( 3 . heart ) ( 4 . heart ) ) )
   ( setf *hand2* '( ( 2 . club ) ( king . diamond ) ( 7 . spade ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   ( report-the-result )
;p1 bust - p2 flush
   ( format t ">>> Testing: report-the-result - p1 bust, p2 flush~%")
   ( setf *hand1* '( ( 6 . spade ) ( 8 . heart ) ( 10 . club ) ) )
   ( setf *hand2* '( ( 6 . diamond ) ( king . diamond ) ( 4 . diamond ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   ( report-the-result )
;p1 flush high - p2 flush
   ( format t ">>> Testing: report-the-result - p1 flush high, p2 flush~%")
   ( setf *hand1* '( ( ace . heart ) ( 2 . heart ) ( 6 . heart ) ) )
   ( setf *hand2* '( ( 4 . spade ) ( 6 . spade) ( 9 . spade ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   ( report-the-result )
;p1 flush - p2 flush high
   ( format t ">>> Testing: report-the-result - p1 flush, p2 flush high~%")
   ( setf *hand1* '( ( 4 . spade ) ( 8 . spade ) ( 9 . spade ) ) )
   ( setf *hand2* '( ( 7 . diamond ) ( jack . diamond ) ( king . diamond ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   ( report-the-result )
;p1 straight flush - p2 flush
   ( format t ">>> Testing: report-the-result - p1 stright flush, p2 flush~%")
   ( setf *hand1* '( ( 10 . diamond ) ( jack . diamond ) ( queen . diamond ) ) )
   ( setf *hand2* '( ( jack . heart ) ( 10 . heart ) ( 6 . heart ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   ( report-the-result )
;p1 flush - p2 straight flush
   ( format t ">>> Testing: report-the-result - p1 flush, p2 straight flush~%")
   ( setf *hand1* '( ( 6 . heart ) ( 2 . heart ) ( 8 . heart ) ) )
   ( setf *hand2* '( ( 9 . diamond ) ( 10 . diamond ) ( jack . diamond ) ) )
   ( analyze-game )
   ( format t "*hand1* = ~A~%" ( write-to-string *hand1* ) )
   ( format t "*hand2* = ~A~%" *hand2* )
   ( format t "*game-state* = ~A~%" *game-state* )
   ( report-the-result )
)
```

Task 6: Play Game

Task 6 Demo:

```
[23]> ( demo--hand-rep )
((9 . CLUB) (5 . CLUB) (10 . HEART)) --> ((9 . C) (5 . C) (10 . H))
NIL
[24]> ( demo--hand-rep )
((8 . HEART) (3 . CLUB) (5 . CLUB)) --> ((8 . H) (3 . C) (5 . C))
NIL
[25]> ( demo--hand-rep )
((9 . DIAMOND) (JACK . DIAMOND) (7 . HEART)) --> ((9 . D) (J . D) (7 . H))
NIL
[26]> ( demo--hand-rep )
((ACE . CLUB) (4 . CLUB) (4 . DIAMOND)) --> ((A . C) (4 . C) (4 . D))
NIL
[27]> ( demo--hand-rep )
((6 . DIAMOND) (10 . SPADE) (ACE . CLUB)) --> ((6 . D) (10 . S) (A . C))
NIL
[28]> ( demo--hand-rep )
((QUEEN . HEART) (KING . HEART) (4 . HEART)) --> ((Q . H) (K . H) (4 . H))
NIL
[29]> ( demo--play-game )
>>> Testing: play-game
((6 . H) (A . C) (Q . C))      ((2 . H) (K . S) (3 . S))
((9 . C) (A . C) (Q . C))      ((5 . D) (K . S) (3 . S))
--> Player 1 wins with (ACE HIGH CLUB FLUSH)
NIL
[30]> ( demo--play-game )
>>> Testing: play-game
((A . D) (Q . D) (Q . C))      ((9 . D) (3 . S) (8 . S))
((A . D) (3 . C) (Q . C))      ((2 . D) (3 . S) (8 . S))
((A . D) (10 . S) (Q . C))      ((2 . D) (9 . H) (8 . S))
((A . D) (7 . D) (Q . C))      ((5 . C) (9 . H) (8 . S))
((A . D) (J . C) (Q . C))      ((J . D) (9 . H) (8 . S))
((A . D) (J . C) (6 . H))      ((7 . S) (9 . H) (8 . S))
((A . D) (J . H) (6 . H))      ((7 . S) (9 . H) (Q . S))
((A . D) (4 . S) (6 . H))      ((7 . S) (Q . H) (Q . S))
((K . D) (4 . S) (6 . H))      ((9 . S) (Q . H) (Q . S))
((K . C) (4 . S) (6 . H))      ((2 . H) (Q . H) (Q . S))
((K . C) (4 . S) (A . S))      ((6 . D) (Q . H) (Q . S))
((K . C) (4 . S) (10 . H))      ((6 . D) (2 . S) (Q . S))
((K . C) (4 . H) (10 . H))      ((6 . D) (2 . S) (6 . S))
((K . C) (4 . H) (A . H))      ((6 . D) (2 . S) (6 . C))
((K . C) (4 . H) (8 . C))      ((2 . C) (2 . S) (6 . C))
((4 . C) (4 . H) (8 . C))      ((2 . C) (2 . S) (8 . H))
((J . S) (4 . H) (8 . C))      ((5 . H) (2 . S) (8 . H))
((7 . C) (4 . H) (8 . C))      ((9 . C) (2 . S) (8 . H))
((7 . C) (4 . H) (10 . D))      ((9 . C) (2 . S) (8 . D))
((7 . C) (A . C) (10 . D))      ((9 . C) (2 . S) (5 . D))
((7 . C) (3 . H) (10 . D))      ((9 . C) (2 . S) (K . S))
((7 . C) (3 . H) (5 . S))      ((9 . C) (2 . S) (K . H))
((7 . C) (3 . H) (4 . D))      ((10 . C) (2 . S) (K . H))
((7 . H) (3 . H) (4 . D))      ((3 . D) (2 . S) (K . H))
--> The game is a draw. The deck is dead.
NIL
```

Task 6 Demo cont:

```
[31]> ( demo--play-game )
>>> Testing: play-game
((J . C) (3 . D) (9 . D))      ((6 . C) (8 . D) (2 . S))
((10 . S) (3 . D) (9 . D))      ((6 . C) (8 . D) (5 . C))
((10 . S) (3 . D) (3 . S))      ((6 . S) (8 . D) (5 . C))
((10 . S) (3 . D) (K . S))      ((6 . S) (8 . D) (J . D))
((10 . C) (3 . D) (K . S))      ((6 . S) (6 . H) (J . D))
((10 . C) (K . H) (K . S))      ((A . H) (6 . H) (J . D))
((10 . C) (K . H) (8 . H))      ((A . H) (6 . H) (2 . D))
((10 . C) (K . H) (7 . D))      ((A . H) (8 . S) (2 . D))
((10 . C) (K . H) (Q . H))      ((A . S) (8 . S) (2 . D))
((10 . C) (K . H) (7 . H))      ((7 . S) (8 . S) (2 . D))
((A . D) (K . H) (7 . H))      ((J . H) (8 . S) (2 . D))
((A . D) (K . H) (3 . C))      ((J . H) (8 . S) (Q . D))
((4 . D) (K . H) (3 . C))      ((J . H) (2 . H) (Q . D))
((2 . C) (K . H) (3 . C))      ((J . H) (A . C) (Q . D))
((2 . C) (6 . D) (3 . C))      ((J . H) (9 . C) (Q . D))
((2 . C) (6 . D) (5 . S))      ((J . H) (J . S) (Q . D))
((9 . H) (6 . D) (5 . S))      ((J . H) (7 . C) (Q . D))
((K . D) (6 . D) (5 . S))      ((J . H) (10 . H) (Q . D))
((K . C) (6 . D) (5 . S))      ((5 . H) (10 . H) (Q . D))
((10 . D) (6 . D) (5 . S))      ((4 . C) (10 . H) (Q . D))
((10 . D) (3 . H) (5 . S))      ((4 . H) (10 . H) (Q . D))
((10 . D) (3 . H) (9 . S))      ((4 . H) (10 . H) (8 . C))
((5 . D) (3 . H) (9 . S))      ((4 . H) (4 . S) (8 . C))
((5 . D) (3 . H) (Q . C))      ((4 . H) (4 . S) (Q . S))
--> The game is a draw. The deck is dead.
NIL
[32]> ( demo--play-game )
>>> Testing: play-game
((9 . D) (5 . D) (8 . C))      ((10 . S) (5 . H) (A . S))
((3 . D) (5 . D) (8 . C))      ((4 . D) (5 . H) (A . S))
((7 . H) (5 . D) (8 . C))      ((4 . D) (4 . S) (A . S))
((4 . C) (5 . D) (8 . C))      ((4 . D) (4 . S) (A . D))
((4 . C) (J . S) (8 . C))      ((A . C) (4 . S) (A . D))
((4 . C) (J . S) (4 . H))      ((7 . D) (4 . S) (A . D))
((4 . C) (J . S) (8 . D))      ((Q . D) (4 . S) (A . D))
((4 . C) (J . S) (7 . C))      ((Q . S) (4 . S) (A . D))
((4 . C) (J . S) (J . D))      ((Q . C) (4 . S) (A . D))
((4 . C) (J . S) (K . H))      ((9 . H) (4 . S) (A . D))
((4 . C) (J . S) (6 . H))      ((8 . H) (4 . S) (A . D))
((4 . C) (J . S) (10 . D))      ((8 . H) (4 . S) (A . H))
((Q . H) (J . S) (10 . D))      ((8 . H) (4 . S) (2 . D))
((Q . H) (J . S) (8 . S))      ((8 . H) (5 . S) (2 . D))
((Q . H) (J . S) (3 . C))      ((5 . C) (5 . S) (2 . D))
((K . D) (J . S) (3 . C))      ((5 . C) (5 . S) (K . C))
((6 . C) (J . S) (3 . C))      ((5 . C) (5 . S) (10 . C))
((6 . C) (2 . S) (3 . C))      ((6 . S) (5 . S) (10 . C))
((6 . C) (3 . H) (3 . C))      ((6 . S) (5 . S) (2 . H))
((6 . C) (3 . H) (6 . D))      ((7 . S) (5 . S) (2 . H))
((6 . C) (3 . S) (6 . D))      ((J . C) (5 . S) (2 . H))
((6 . C) (9 . C) (6 . D))      ((9 . S) (5 . S) (2 . H))
((6 . C) (9 . C) (2 . C))      ((9 . S) (J . H) (2 . H))
--> Player 1 wins with (9 HIGH CLUB FLUSH)
NIL
```

```
[33]> ( demo--play-game )
>>> Testing: play-game
((10 . H) (4 . D) (9 . H))      ((5 . H) (K . D) (7 . D))
((4 . H) (4 . D) (9 . H))      ((5 . H) (K . D) (J . D))
((6 . H) (4 . D) (9 . H))      ((5 . H) (7 . S) (J . D))
((6 . H) (4 . D) (Q . H))      ((5 . H) (5 . D) (J . D))
((6 . H) (4 . D) (6 . C))      ((K . H) (5 . D) (J . D))
((6 . H) (4 . D) (2 . S))      ((A . D) (5 . D) (J . D))
--> Player 2 wins with (ACE HIGH DIAMOND FLUSH)
NIL
[34]> ( demo--play-game )
>>> Testing: play-game
((Q . S) (Q . D) (2 . H))      ((9 . S) (A . S) (4 . D))
((Q . S) (Q . D) (7 . H))      ((2 . D) (A . S) (4 . D))
((Q . S) (7 . S) (7 . H))      ((9 . C) (A . S) (4 . D))
((Q . S) (7 . S) (3 . C))      ((9 . C) (A . H) (4 . D))
((Q . S) (7 . S) (Q . C))      ((9 . C) (5 . C) (4 . D))
((Q . S) (7 . S) (3 . S))      ((9 . C) (A . C) (4 . D))
--> Player 1 wins with (QUEEN HIGH SPADE FLUSH)
NIL
[35]> ( demo--play-game )
>>> Testing: play-game
((K . C) (A . C) (7 . S))      ((6 . C) (4 . S) (K . H))
((K . C) (8 . C) (7 . S))      ((3 . D) (4 . S) (K . H))
((K . C) (A . H) (7 . S))      ((Q . H) (4 . S) (K . H))
((2 . D) (A . H) (7 . S))      ((Q . H) (9 . S) (K . H))
((2 . D) (8 . S) (7 . S))      ((Q . H) (3 . S) (K . H))
((2 . D) (8 . S) (9 . C))      ((J . C) (3 . S) (K . H))
((2 . D) (A . S) (9 . C))      ((K . D) (3 . S) (K . H))
((2 . D) (A . S) (J . D))      ((10 . H) (3 . S) (K . H))
((2 . D) (A . S) (7 . C))      ((10 . H) (3 . S) (2 . S))
((10 . C) (A . S) (7 . C))      ((10 . H) (3 . S) (7 . D))
((10 . C) (A . S) (8 . H))      ((10 . H) (6 . D) (7 . D))
((J . S) (A . S) (8 . H))      ((10 . H) (A . D) (7 . D))
((J . S) (2 . C) (8 . H))      ((10 . H) (6 . H) (7 . D))
((J . S) (4 . C) (8 . H))      ((2 . H) (6 . H) (7 . D))
((Q . C) (4 . C) (8 . H))      ((5 . H) (6 . H) (7 . D))
((K . S) (4 . C) (8 . H))      ((5 . H) (6 . H) (4 . H))
--> Player 2 wins with (6 HIGH HEART FLUSH)
NIL
```

Task 6 Code:

```
( defun hand-rep ( hand )
   ( list ( new-rep ( first hand ) ) ( new-rep ( second hand ) )
      ( new-rep ( third hand ) ) )
)

( defun new-rep ( card )
   ( cond
      ( ( equal ( car card ) 'ace ) ( setf rep 'a ) )
      ( ( equal ( car card ) 'king ) ( setf rep 'k ) )
      ( ( equal ( car card ) 'queen ) ( setf rep 'q ) )
      ( ( equal ( car card ) 'jack ) ( setf rep 'j ) )
      ( t
         ( setf rep ( car card ) ) )
   )
   ( cond
      ( ( equal ( cdr card ) 'diamond ) ( setf suit 'd ) )
      ( ( equal ( cdr card ) 'heart ) ( setf suit 'h ) )
      ( ( equal ( cdr card ) 'spade ) ( setf suit 's ) )
      ( t
         ( setf suit 'c )
      )
   )
   ( cons rep suit )
)

( defun demo--hand-rep ( &aux hand )
   ( establish-shuffled-deck )
   ( setf internal ( list ( pop *deck*) ( pop *deck*) ( pop *deck*) ) )
   ( setf external ( hand-rep internal ) )
   ( format t "~A --> ~A~%" internal external )
   nil
)

( defun play-game ()
   ( increment '*game-count* )
   ( deal-hands )
   ( make-moves )
   ( report-the-result )
)
```

Task 6 Code cont:

```
( defun make-moves ()
    ( increment '*turn-count* )
    ( format t "~A      ~A~%" ( hand-rep *hand1* ) ( hand-rep *hand2* ) )
    ( if ( not ( game-over-p ) )
        (let ()
            ( players-each-take-a-turn )
            ( make-moves )
        )
    )
    nil
)

( defun game-over-p ()
    ( analyze-game )
    ( or
        ( not ( equal *game-state* '( bust bust ) ) )
        ( null *deck* )
    )
)

( defun demo--play-game ()
    ( format t ">>> Testing: play-game~%" )
    ( play-game )
)
```

Task 7: Computing Statistics

Task 7 Demo:

```
*game-count* = 10
*turn-count* = 111
*win1-count* = 5
*win2-count* = 3
*draw-count* = 1
*f1f2-count* = 0
NIL
```

```
*game-count* = 100
*turn-count* = 1185
*win1-count* = 43
*win2-count* = 40
*draw-count* = 17
*f1f2-count* = 1
NIL
```

```
*game-count* = 1000
*turn-count* = 9786
*win1-count* = 436
*win2-count* = 450
*draw-count* = 109
*f1f2-count* = 15
NIL
```

Task 7 Code:

```
; Counter initialization -- initialize once so the game can be played regardless of
; whether or not statistics are being computed

( defun init-counters ()
   ( setf *win1-count* 0 )
   ( setf *win2-count* 0 )
   ( setf *draw-count* 0 )
   ( setf *turn-count* 0 )
   ( setf *f1f2-count* 0 )
   ( setf *game-count* 0 )
   nil
)

( init-counters )

; Flexible counter incrementation
( defun increment (name)
   ( set name ( + ( eval name ) 1 ) )nil
)

; The main statistics computation program
( defun compute-statistics ( n )
   ( init-counters )
   ( play-game-n-times n )
   ( format t "*game-count* = ~A~%" *game-count* )
   ( format t "*turn-count* = ~A~%" *turn-count* )
   ( format t "*win1-count* = ~A~%" *win1-count* )
   ( format t "*win2-count* = ~A~%" *win2-count* )
   ( format t "*draw-count* = ~A~%" *draw-count* )
   ( format t "*f1f2-count* = ~A~%" *f1f2-count* )
   nil
)

; Program to play the game n times
( defun play-game-n-times ( n )
   ( cond
      ( ( > n 0 )
        ( play-game )
        ( play-game-n-times ( - n 1 ) )
      )
   )
)
```

Task 8: The Heuristic Player

Task 8 Demo:

```
[3]> ( demo--randomly-heuristically-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((4 . CLUB) (JACK . SPADE) (3 . DIAMOND))
--- *hand2* = ((QUEEN . HEART) (7 . HEART) (9 . HEART))
--- *hand1* = (NIL (JACK . SPADE) (3 . DIAMOND))
--- *hand2* = ((QUEEN . HEART) (7 . HEART) NIL)
NIL
[4]> ( demo--randomly-heuristically-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((QUEEN . HEART) (9 . HEART) (10 . SPADE))
--- *hand2* = ((8 . SPADE) (KING . CLUB) (8 . CLUB))
--- *hand1* = ((QUEEN . HEART) (9 . HEART) NIL)
--- *hand2* = (NIL (KING . CLUB) (8 . CLUB))
NIL
[5]> ( demo--randomly-heuristically-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((KING . SPADE) (4 . SPADE) (8 . HEART))
--- *hand2* = ((10 . SPADE) (7 . CLUB) (KING . DIAMOND))
--- *hand1* = ((KING . SPADE) (4 . SPADE) NIL)
--- *hand2* = ((10 . SPADE) (7 . CLUB) NIL)
NIL
[6]> ( demo--randomly-heuristically-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((JACK . HEART) (JACK . CLUB) (9 . SPADE))
--- *hand2* = ((8 . HEART) (QUEEN . HEART) (6 . SPADE))
--- *hand1* = (NIL (JACK . CLUB) (9 . SPADE))
--- *hand2* = ((8 . HEART) (QUEEN . HEART) NIL)
NIL
[7]> ( demo--randomly-heuristically-discard-cards )
>>> Testing: randomly-discard-cards
--- *hand1* = ((2 . SPADE) (6 . SPADE) (7 . HEART))
--- *hand2* = ((3 . DIAMOND) (ACE . HEART) (2 . CLUB))
--- *hand1* = ((2 . SPADE) (6 . SPADE) NIL)
--- *hand2* = ((3 . DIAMOND) NIL (2 . CLUB))
NIL
```

Task 8 Demo cont:

```
[8]> ( play-game )
((8 . H) (A . H) (4 . D))      ((2 . H) (Q . D) (J . S))
((8 . H) (Q . S) (4 . D))      ((2 . H) (K . S) (J . S))
((8 . H) (2 . C) (4 . D))      ((K . H) (K . S) (J . S))
((8 . H) (4 . C) (4 . D))      ((A . S) (K . S) (J . S))
--> Player 2 wins with (ACE HIGH SPADE FLUSH)
NIL
[9]> ( play-game )
((6 . S) (9 . D) (A . H))      ((10 . S) (K . C) (9 . C))
((Q . C) (9 . D) (A . H))      ((8 . S) (K . C) (9 . C))
((Q . C) (9 . D) (2 . H))      ((7 . D) (K . C) (9 . C))
((Q . C) (J . D) (2 . H))      ((3 . H) (K . C) (9 . C))
((Q . C) (J . D) (10 . C))      ((8 . C) (K . C) (9 . C))
--> Player 2 wins with (KING HIGH CLUB FLUSH)
NIL
[10]> ( play-game )
((A . S) (2 . D) (3 . H))      ((2 . C) (10 . C) (J . D))
((A . S) (2 . D) (8 . S))      ((2 . C) (10 . C) (10 . D))
((A . S) (2 . D) (8 . H))      ((2 . C) (10 . C) (K . S))
((A . S) (2 . D) (9 . D))      ((2 . C) (10 . C) (9 . S))
((A . S) (Q . D) (9 . D))      ((2 . C) (10 . C) (4 . S))
((A . S) (2 . S) (9 . D))      ((2 . C) (10 . C) (J . C))
--> Player 2 wins with (JACK HIGH CLUB FLUSH)
NIL
[11]> ( play-game )
((10 . C) (K . C) (2 . H))      ((2 . D) (J . C) (9 . D))
((3 . H) (K . C) (2 . H))      ((2 . D) (7 . C) (9 . D))
((3 . H) (K . D) (2 . H))      ((2 . D) (Q . H) (9 . D))
((3 . H) (K . D) (6 . D))      ((2 . D) (8 . C) (9 . D))
((3 . H) (K . D) (K . H))      ((2 . D) (Q . C) (9 . D))
((3 . H) (K . D) (10 . S))      ((2 . D) (8 . S) (9 . D))
((3 . H) (K . D) (Q . D))      ((2 . D) (J . H) (9 . D))
((3 . H) (J . D) (Q . D))      ((2 . D) (5 . S) (9 . D))
((8 . H) (J . D) (Q . D))      ((2 . D) (7 . H) (9 . D))
((8 . H) (J . D) (9 . H))      ((2 . D) (2 . C) (9 . D))
((8 . H) (J . D) (K . S))      ((2 . D) (A . D) (9 . D))
--> Player 2 wins with (ACE HIGH DIAMOND FLUSH)
NIL
```

```
*game-count* = 10
*turn-count* = 47
*win1-count* = 1
*win2-count* = 9
*draw-count* = 0
*f1f2-count* = 1
NIL
```

```
*game-count* = 100
*turn-count* = 483
*win1-count* = 24
*win2-count* = 76
*draw-count* = 0
*f1f2-count* = 1
NIL
```

```
*game-count* = 1000
*turn-count* = 4936
*win1-count* = 209
*win2-count* = 789
*draw-count* = 0
*f1f2-count* = 36
NIL
```

Task 8 Code:

```
( defun players-each-take-a-turn ()
   ( randomly-heuristically-discard-cards )
   ( replace-cards )
   nil
)
```

```
( defun randomly-heuristically-discard-cards ()
   ( randomly-discard-card-from-hand1 )
   ( heuristic-discard-card-from-hand2 )
)

( defun heuristic-discard-card-from-hand2 ()
   ( setf card1 ( cdr ( first *hand2* ) ) )
   ( setf card2 ( cdr ( second *hand2* ) ) )
   ( setf card3 ( cdr ( third *hand2* ) ) )
   ( cond
      ( ( equal card1 card2 )
         ( setf ( nth 2 *hand2* ) nil )
      )
      ( ( equal card1 card3 )
         ( setf ( nth 1 *hand2* ) nil )
      )
      ( ( equal card2 card3 )
         ( setf ( nth 0 *hand2* ) nil )
      )
      (t
         (randomly-discard-card-from-hand2 )
      )
   )
)

( defun demo--randomly-heuristically-discard-cards ()
   ( format t ">>> Testing: randomly-discard-cards~%" )
   ( deal-hands )
   ( format t "--- *hand1* = ~A~%" *hand1* )
   ( format t "--- *hand2* = ~A~%" *hand2* )
   ( randomly-heuristically-discard-cards )
   ( format t "--- *hand1* = ~A~%" *hand1* )
   ( format t "--- *hand2* = ~A~%" *hand2* )
   nil
)
```